Research Journal of Recent Sciences _____ ISSN 2277-2502
Vol. **2(8),** 31-39, August **(2013)** *Res.J.Recent Sci.*

Packet Classification Algorithm Based on Geometric Tree by using Recursive Dimensional Cutting (DimCut)

Hediyeh Amir Jahanshahi Sistani[1], **Sayyed Mehdi Poustchi Amin**[1] **and Haridas Acharya**[2]
[1]Department of Computer Studies and Research, Symbiosis International University, Pune, INDIA
[2]Allana Institute of Management Science, Pune University, Pune, INDIA

Available online at: www.isca.in
Received 26th February 2013, revised 10th March 2013, accepted 5th April 2013

Abstract

The Packet classification is a key function of the firewalls and routers. Internet firewalls, routers and service providers perform different operations at different flows. With the increasing demands on router performance, there is a need for algorithms that can classify packets quickly with minimal storage requirements. This paper presents a heuristic, called Packet Classification Algorithm Based on Geometric Tree by using Recursive Dimensional Cutting (DimCut), which exploits the structure found in classifiers. It, like the previously well-known algorithm, HiCutsis based on a decision tree structure. After examinationDimCut algorithm, to classify packets based on five header fields, it is found that the algorithm can classify packets quickly. Our proposal extends HiCuts with new heuristics ideas and new implementing techniques while retaining HiCuts' basic framework. The DimCut algorithm has two separated levels, pre-processinglevel (tree construction and making index table) and search level. The algorithm provides a full description of the chief data structures and tuneable parameters. We explain details that describe the pre-processing algorithm and the search process, also provide the source code that permits the actual implementation.

Keywords: Packet classification, firewalls, routers, heuristics, dimensional cutting, rules.

Introduction

Network routers can provide advanced network services after being enabled by packet classification. Rather than the basic packet forwarding, this also includes network security, policy-based routing, and quality of service (QoS) assurance. High performance algorithms are of great interest to academics and industrialists. Large scale packet classification has become driving factors of network security and QoS. Firewalls also have to classify packets, where speed of decision making to deny or not to deny, is of utmost importance.

The basic problems of packet classification are huge rule sets (size of rule set), increasing network traffic (traffic intensity) and large dimensionality of the packet attributes data base (large item sets)[1-7].

In this paper we have proposed improvements over existing nonlinear type of classification algorithms, in particular the ones which use HiCuts[8]. The improvements have been validated through simulated trials. In brief, we contribute to the proposal to examine new heuristic methods of packet classification that have good average case performance and utilize storage reasonably and provide benchmark results on the practical performance of our proposed algorithms.

The paper is organized as follows. Section 1 surveys from high level perspective the existing algorithms to capture the basic ideas, as each approach has its own advantages and disadvantages with reference to throughput, cost, facility of implementation and scalability. Section 2 deals with details of our algorithm. Section 3 evaluates issues of implementation. Section 4 presents the results of experiments. Section 5 takes a comparative approach to related work. We conclude in section 6.

Packet Classification Problem

Marking a packet to allow or disallow is called the process of classification. A set of policy rules control the acceptance or denial of packets. A packet classifier must compare header fields of every incoming packet against a set of rules in order to assign a flow identifier that is applied insecurity policies[8,9,10].

A rule must specify a set of headers and the policy to be in use. We may define formally a rule set as:

Definition: A Rule Space is a collection of rules, specified as a table (flat data base) with, Columns as RuleID, 'D' header field specification as H_1, H_2… H_d, and an Action column. Each record (Row) specifies a rule. Number D would be the Dimension of the rule space.

Process of classification requires, applying the rules from the top. Attributes of the packet to be classified are matched with values in the Header columns, and if successful the Action becomes applicable.

For example, a typical Iptables rule may specify: A INPUT -p tcp --dport 22 -j ACCEPT

This rule is to be interpreted as "a TCP packet is to be accepted if the destination port is equal to 22 ". If no match is found the next rule in the table is to be tested. In general the process continues sequentially till a match is found. Of course this approach would be a brute force method and the complexity will be the worst.

If n_1, n2… nk are the possible distinct values columns can take then the complete Rule space will containrows in all.

$\mathbf{N} = (n_1\ n2\ \ldots.\ n_D) = \pi(n_i)$
Here the operator π stands for the product. Here we have every possible value of this would mean any linear search would require complexity of O (N), which is not desirable. Hence there has been a need for design of better algorithms, and this has been a topic of interest among researchers. For H1 = source IPV4, H2 = destination IPV4, H3= Source Port No, H4= destination Port No and H5 = 4 possible choices for protocol, N would amount to, 2^{32}x 2^{32} x 2^{16}x2^{16} x 2^2= 2^{98}~ 88K combinations to search from. In practice all possible values which a header may take are not present in the rule base.

Background: Algorithms for packet classification, found in literature, are of four types. Typification is according to their basic design and the process of classification adopted[11]. The four types are: The Exhaustive Search: brute force method and obviously is the most inefficient. Decision Tree: prior construction decision tree are followed by enhanced searches. Decomposition: decompose the multiple field searches into instances of single field searches, perform independent searches on each packet field, and then combine the results. Tuple Space: partition the filter set according to the number of specified bits in the filters, probe the partitions or a subset of the partitions using simple exact match searches.

When alternatives are designed testing becomes an issue. Performance metrics for testing such algorithms would include: {Search speed, Storage requirements, Fast updates, Scalability, Flexibility}.

A classification algorithm should support general rules, including prefixes, range, exact value and wildcards. Providing better data structures to Rule Bases, assigning priorities to rules to avoid conflicting and multiple matches, and pre processing the Rule Base, have been some of the common techniques used by various researchers to improve the algorithms[11]. HiCuts is an example of decision tree-based packet classification algorithm. It takes the geometric view of the packet classification problem. Since it forms the basis for our new algorithm, we discuss these in brief in the next step.

HiCuts: Gupta and McKeown introduced a seminal technique called Hierarchical Intelligent Cuttings (HiCuts)[8]. Each rule in the rule set defines a d-dimensional rectangle in d-dimensional space, where d is the number of fields in the rule. The algorithm preprocesses the rule set to build a decision tree with leaves containing a subset of rules with number of rules bound by a prescribed threshold. Packet header fields must traverse the decision tree until a leaf is reached, i.e. a subset is identified for further processing. The rules from the leaf subsets are then linearly searched for a match.

A simple illustrative example[1]: Figure1 has a Rule set of five rules {R1, R2, R3, R4, R5} represented in a two dimensional space. Here N = 5, D = 2. The illustration assumes a threshold of t=2, which is the maximum number of rules a subset at leaf node can contain.

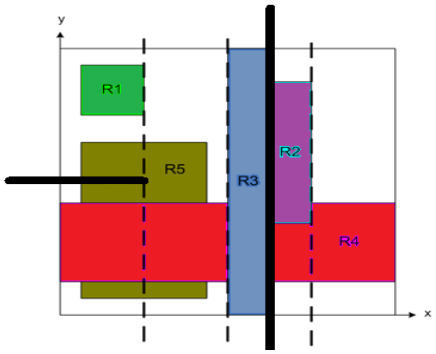

Figure-1
The rule set has two fields, each rectangle represents a rule and 5 rules are shown geometrically

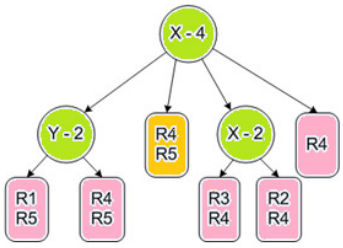

Figure-2
A geometric view of a tree constructed according to cutting of the rule set

The root node covers all portions of the d dimensional space. At the top level, whole space is cut along the x-axis (vertical cuts) to generate 4 equal sized sub-regions. One can see from figure 2 that this has resulted in to two leaf nodes, since they have rules less than or equal to t. In the next step, two of the remaining sub-regions are chosen to cut further. The left most node has a horizontal cut, whereas the other one has a vertical cut again (x-axis). At the first level there were four giving us two leaf nodes, and two nodes which required further cuts. When the second level cut is implemented all the subsets have rules less than t, meeting the threshold requirement. A cut is implemented at a node only if the corresponding subset has rules more than the threshold (t).

HiCuts utilizes four heuristics for optimizing the tree. The first two, at each node select the appropriate dimension to cut and the number of cuts. The last two, reduce the storage requirement and eliminate redundancy in the tree.

Issues with HiCuts are that a larger number of cuts at a node reduce the tree depth, but may increase rule replication and also the number of children, which may not achieve good rule separation. The second heuristic attempts to maximize the number of cuts, and hence minimize the depth, while limiting the total number of rules at all the nodal children with a factor, called space factor. The redundant sibling nodes which share an identical set of rules are the third heuristic target. This one merges such siblings into one node. Another kind of redundancy exists when a higher-priority rule overlaps a lower-priority rule completely within a node's subspace. In which case, no packet would ever match the lower-priority rule which can be removed, as done by the fourth heuristic[12].

The HiCuts has specifications such as scalability, low memory consumption and reasonable speed, to make it one of the most efficient packet classification algorithms. But choosing suitable point for making cut in the intended dimension is what that the algorithm designers haven't worked on it[9].

New Algorithm

Choosing suitable point for making cut, in HiCuts algorithms still seems to be a problem that algorithm designers haven't worked sufficiently on it[9]. Our algorithm adds some modifications and improvements on the HiCuts algorithm, developed by Gupta and McKeown.Consider the following definitions:

Definition: Let wc(H) be the count of wild card entries in the column H in the whole of the rule set.

Definition: Let gd(H) be the geometric distance associated with column H in the whole of the rule set.

We make use of these values associated with every field in the rule set to define appropriate heuristics for the purpose of selecting the dimensions for the cuts. We follow these guide lines and principles: i. Dimension Selection: Select the two fields Ha, Hb which have the least wc( ) values, as the two selected dimensions or alternatively we select Ha, Hb which have least gd( ) values. ii. Number of cuts and Bucket size: Compute the number of cuts as the number of cuts, and the bucket size threshold as: NC1 = [20 + (N/1000)] = Number of cuts, B = [N/ (20 + (N/1000))] = Bucket size (The threshold). Here N = Total Number of rules, in the complete rule set, iii. Separate those rules in the same chosen field as cut dimension which have wildcard value and shift them to the bucket and reject their use for making the decision tree. iv. Building index tables to facilitate search within: Build an index table for each bucket. v. The number of cuts has to be decided at the first time cutting and also the bucket size threshold of the algorithm has to be identified, with the purpose of trying to avoid splitting of rules while cutting. vi. Recursion: The best dimension for next cut level is identified after the first cut, again using the same principles. vii. New algorithm has two separate levels, pre-processing level (tree construction and making index table) and search level. viii. Use the Link list data structure at the input stage and work on large rule sets.

The new algorithm provides a full description of the chief data structures and tunable parameters. We explain details that describe the preprocessing algorithm and the search process and provide the source code that permits the actual implementation.

Packet classification by Decision Tree Algorithm is to construct a decision tree where the leaves of the tree contain rules or subsets of rules. In order to perform a search using a decision tree, one should construct a search key from the packet header fields and traverse the decision tree by using individual bits or subsets of bits from the search key. The search continues until reaching a leaf node storing the best matching rule. A node becomes a leaf of the tree when it has few threshold rules.

To build a decision tree data structure, the algorithm carefully preprocessesthe rules. The decision tree is checked for a leaf node every time a packet arrives, as it stores small set of rules and when the search tree is constructed the choice about the shape and depth of the decision tree and local decision are made. As far as possible minimum repetition of rules inside the bucketsis adhered to by the preprocessing algorithm that uses a heuristic for distributing rules inside a bucket.

It's shown how to classify rules in figure 3, 4, 5, and 6, in this example rules onlyhave two fields (source Ip address and destination Ip address), each field has 4 bits only.

Priority	Source Add.	Destination Add.
R1	1010	*
R2	1100	1010
R3	0101	1001
R4	*	10*
R5	111*	11*
R6	001*	1100
R7	*	00*
R8	0*	10*
R9	0110	011*
R10	1*	11*

Figure-3
In this example rules are shown in priority that, have two fields (source Ip address and destination Ip address), each field only made of 4 bits

Optimize the decision tree: The decision tree has been constructed by new algorithm. Now, to optimize the decision tree will do according existent methods[8,9]: Eliminating the empty nodes, merging the nodes that are associated with the same set of rules, in case the region covered by the rules, is smaller than the overall size of the region governing the node, one shrinks the region associated with the node to minimum cover, and if the same rule repeated in all nodes in the same level, then separate that rule and make a bucket of that for use at the time of search figure 6. Set the default action for those entry packets that do not match with any bucket. All the rules in all

the buckets should be sorted by priority. First step of pre processing ends with tree construction and bucket making of rules, figure 7.

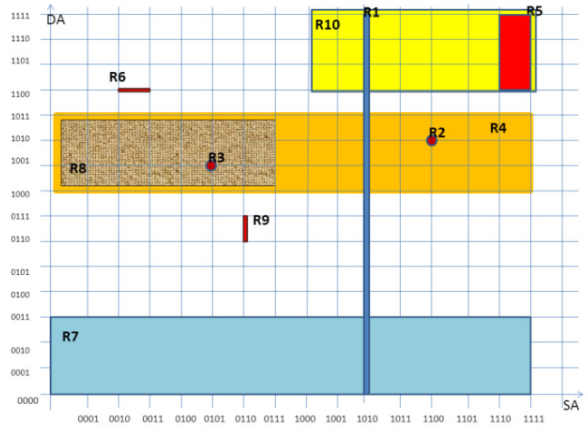

Figure-4
Each rule has two fields, each rectangle represents a rule and 10 rules are shown in geometrically view

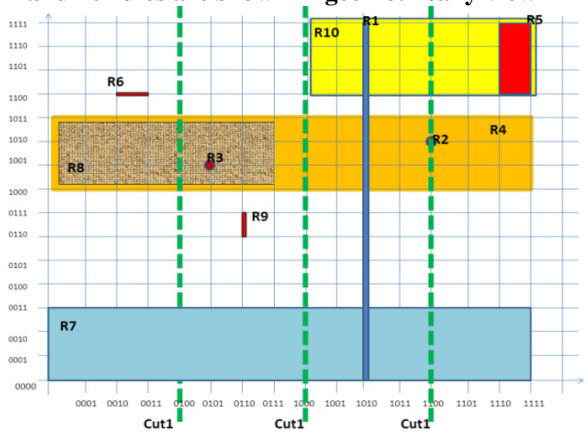

Figure-5
Cutting the geometric view of rule set and decompose the rule set space into small rules buckets

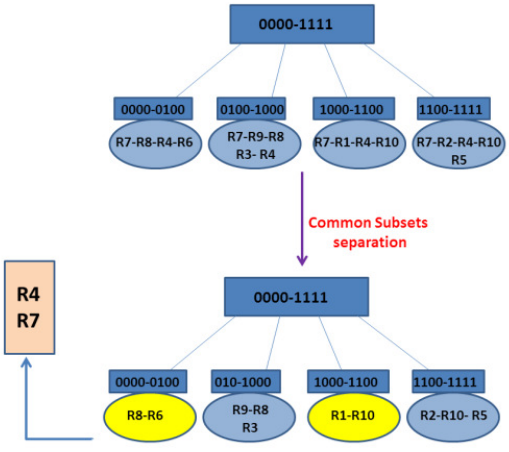

Figure-6
Tree is constructed according cutting the geometric view of rule set

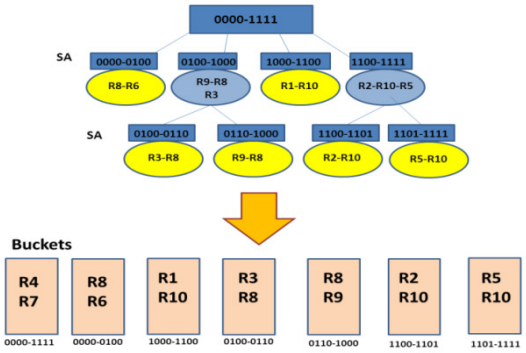

Figure-7
Tree construction and bucket making of rules after optimization

Index table making: For the field that is chosen for cut dimension in each bucket will make an index table. The framework will contain two stages: an index table and rule buckets.

Use the same field of the input packet to search in the index table. If the specific field matches, the matching filter will be selected out of the set in the bucket via linear search (using smaller set of rules). So, ends the preprocessing level and partitioning rules.

All incoming packets need to check at the fields selected during preprocessing. The decision tree traverses to find the buckets that cover the incoming packet. There is priority sorting of all rules. When first match index is found a packet will traverse all regions of possible belonging. The packet will check the all header fields of governing rules linearly. The most prioritized packet is picked up via those that match completely. So the final action (Accept/Deny) will be taken for that incoming packet and the search will end.

It supports incremental update but in case of significant decreasing performance it needs reconstruction. Updating will work in the same manner as the search algorithm. For firewalls a very slow update rate would suffice and entries can be added manually or infrequently.

The Briefed Preprocessing Algorithm: Preprocessing part: i. Read rules and create a link list to store them, ii. Find the cut dimension by using any of 2 heuristics (any dimension that has the smallest geometric length/ any dimension that has the smallest number of wildcards), iii. Calculate the number of cuts by using of (NC= [20+(Number of rules/1000)]) and Calculate the Threshold T= [(Number of rules)/NC], iv. Separate those rules that has wildcard value in the same chosen field as cut dimension in the bucket, v. Construct the tree, For i=1 to NC do, Create buckets (nodes), Assign the rules that covered by buckets (nodes) region , If the number of rules in bucket > Threshold, Split buckets(nodes), Create the index table for rules in buckets, Optimize and compress the tree, END.

The Briefed Search Algorithm: i. Use Search part: Read Packets, For each Packet: Find the buckets that cover the packet, Search in the related index table of those buckets, Find the specific matched rules, Select the higher priority one as a target, Act as its action, iii. End

Experimental Methodology

We have implemented the common linear algorithm (L) and the current algorithms which we call NewHicut2D and NewHicut3D in language C, (GCC 4.4). The codes were compiled with the Code::Blocks 10.05, which is a full-featured IDE all on an Intel Pentium processor 2.00 GHz with RAM 2.00 GB, 32-bit OS, running Microsoft Windows 7 Ultimate and the Oracle VM Virtual Box, so virtual environment were used for all tests. The plan of the simulated experimented was as shown in figure 8.

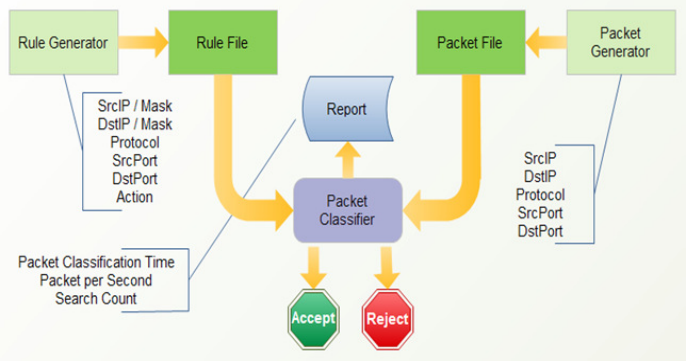

Figure-8
Simulated experimented methodology model

Experimental Results

By comparing the linear with the new algorithm, and using data analysis and graphs, it was possible to prove the proposed algorithm based on decision tree makes packet classification fast. Search performance is evaluated by running through it large number of random packets that obey the weight specifications. We used worst case scenario to do the best evaluation.

Note that our reference implementations are only for the purpose of simulation and evaluation; thus, the source code is not optimized for software. The ambiguity in algorithm details makes the algorithm evaluation and comparison a bit difficult. In our implementation, we allow users to freely configure the parameters. To compare the algorithms under a particular rule set, we choose the configurations that lead to the best overall performance.

Analysis Data collection: The statistical package SPSS was used for Normality test (Kolmogorov–Smirnov), Paired sample T Test. It is duly cautioned that our models show results of average case performance.

Kolmogorov–Smirnov test is testing data for normality of the distribution. It shows our samples are standardized by comparison with a standard normal distribution. The significant score is greater than 0.05, then the data significantly deviate from a normal, so our data is normal as showed in table 1. So according to the Normal data, one can decide about which statistical tests need to be done.

The implementation algorithms constituted of:

3 Types of packet sets generated:	Namely RandomSet / Controlled number of repeats/ worst case scenario.	
Headers constituting the rules:	{Source IP and Destination IP (Exact/prefix),	
	Source Port and Destination Port (Exact value, any, ranges) and	
	Protocol (TCP, UDP, ICMP, ANY)}	
AndActions:	(Accept; Deny, Log, Forward, Nothing)	
Algorithms implemented:	L	(Common linear Algorithm)
	Tree2D	(NewHiCuts2D or DimCut2D)
	Tree3D	(NewHiCuts3D or DimCut3D)

The observation recorded constituted of:

1. The Packet Classification Time: LPC,Tree2DPC,Tree3DPC
2. PPS (Number of Packet Per Second Classification)LPPS,Tree2DPPS,Tree3DPPS
3. Number of Search till packet classification done LSearch,Tree2DSearch,Tree3DSearch
4. The Preprocessing Time/Tree Construction Time: Tree2DSearch,Tree3DSearch
5. Other parameters were:The number of buckets (leaves)/Number of Cuts, Depth of the tree structure, Threshold/Bucket size
6. The number of rules 100 … 100000, and varied threshold sizes.
7. Every test is repeated 3 times to arrive at the average amount in the final result. Then the data is collected and the statistical software SPSS analyzed it.

The t-test is a statistical method that compares the significant difference between the means of two groups of data. The dependent sample t-test (paired sample T Test) enables us to compare packet classification time, number of packet per second, search number between linear and tree attempts. In this test as the significant score is 0.000 (which is less than 0.05) as showed in table 2; therefore there is a significant difference between the means of observed values from L (linear) and Tree2D (NewHighCuts2D) and Tree3D (NewHighCuts3D). Mean times show that times are the least for Tree3D algorithms (table 1).

Figures (Graphs): In all figures (graph) the L means linear search algorithm, the 2DTree means new algorithm by using two fields of rules to construct tree and the 3D Tree means the new algorithm by using three fields of rules to construct tree and the incoming packets are fixed to 10000 numbers at worst case condition that means, force the incoming packets to trace till the end of tree, by setting the Protocol field to X value so the incoming packet doesn't match with any of the rules.

Three of the following figures (graphs) show dependency of PC time (figure 9), PPS (figure 10) and search times (figure 11) respectively on increasing number of rules (X- axis), with 20 % rules repeating once in the rule set and 10% rules repeating twice in the rule set. In all cases the Number of cuts is set as [20 + (R/1000)] and the Bucket size threshold is set as [R/ Number of cuts].

We see that, as the rules number increases, PC time required increases, Packet per Second processing reduces (quickly,the Y axis is on log scale) and vary widely across the tree algorithms. In case of Tree algorithms' number of Packet per Second processing seem to be stabilizing as rulesincreases. However the newer algorithms seem to be more efficient than the linear one. Search time is far lower in case the proposed algorithms compared linear algorithm as can be seen from figure 11.

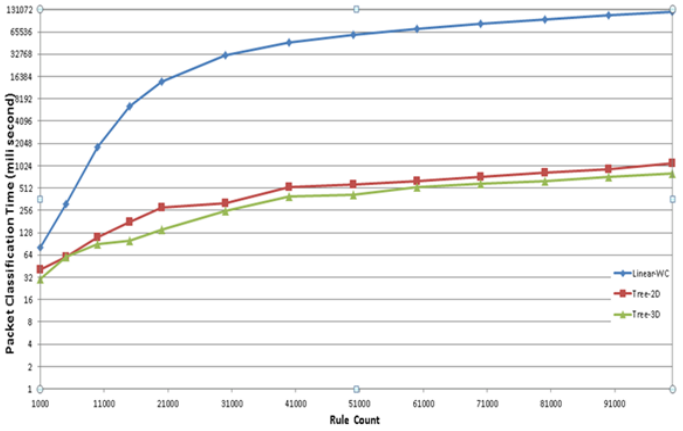

Figure-9
A comparison between the linear and the Non Linear (2D, 3D) algorithm, as we increase rules to measure the packet classification time (micro second)

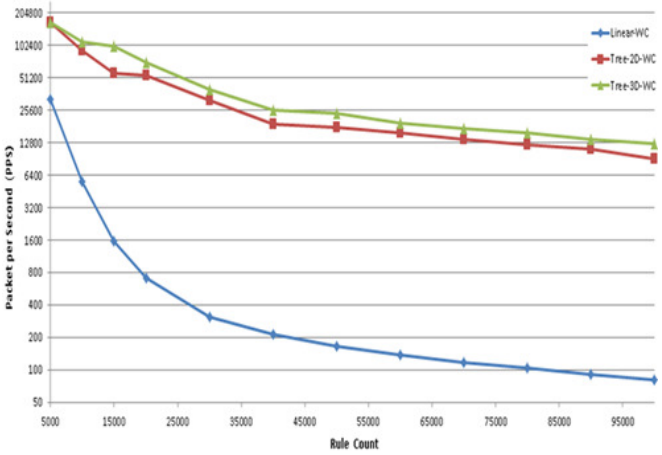

Figure-10
A comparison between the linear and the Non Linear (2D, 3D) algorithm, as we increase rules to measure the number of packet per second processing

Table-1
Shows the data significantly deviate from a normal distribution

One-Sample Kolmogorov-Smirnov	Rules	LPC	Tree2D.PC	Tree3D.PC	LPPS	Tree 2.DPP	Tree3.DPPS	LSearch	Tree2.DSearch	Tree3.DSearch
N	13	13	13	13	13	13	13	13	13	13
Normal Parameters Mean	43923.08	49838.587	482.88492	364.90423	12771.92	57438.62	73103.46	439230769.23	32632028.69	30242754.00
Std. Deviation	33752.683	44356.33781	347.514837	271.312217	34797.713	72949.864	91582.243	337526827.044	23576611.137	22539806.842
Kolmogorov-Smirnov Z	.524	.632	.523	.656	1.546	.991	.922	.524	.497	.505
Asymp. Sig. (2-tailed)	.946	.819	.947	.783	.017	.280	.363	.946	.966	.961

Table-2
Shows significant difference between the means of the linear and Tree attempts

Paired Samples Correlations		N	Correlation	Sig.
Pair 1	L.PC & Tree2D.PC	13	.992	.000
Pair 2	L.PC & Tree3D.PC	13	.998	.000
Pair 3	L.PPS & Tree2D.PPS	13	.907	.000
Pair 4	L.PPS & Tree3D.PPS	13	.931	.000
Pair 5	L.Search& Tree2D.Search	13	.999	.000
Pair 6	L.Search& Tree3D.Search	13	1.000	.000

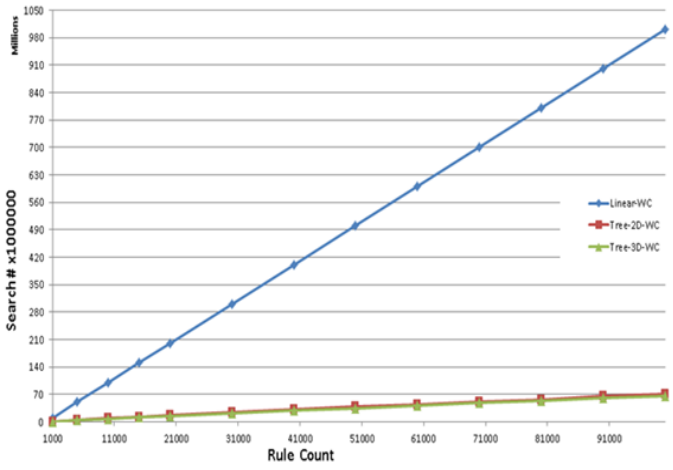

Figure-11
A comparison between the linear and the Non Linear (2D, 3D) algorithm as we increase rules to measure the number of search times

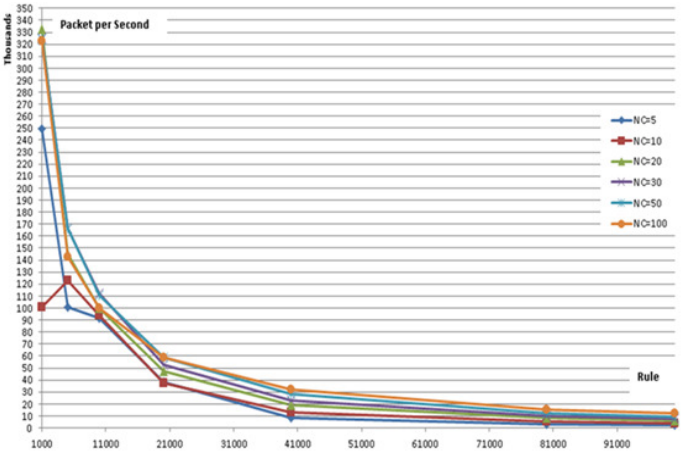

Figure-12
A comparison between the different numbers of cuts for 3DTree algorithm as we increase rules to measure the number of packet per second processing

With incoming packets (fixed to 10000 numbers) at worst case condition and varying number cuts the behavior found is shown in figure 13. The Y axis shows the number of packet per Second processed in log scale. Line graphs correspond to NC =5, 10, 2, 30, 50,100 and the Bucket size threshold is set as (R/ Number of cuts).In contrast, points that match with the NC = [20 + (R/1000)], almost shows higher number of Packet per Second processing across increases in rules. The Figure 12, shows set NC to [20 + (R/1000)], gives us better results.

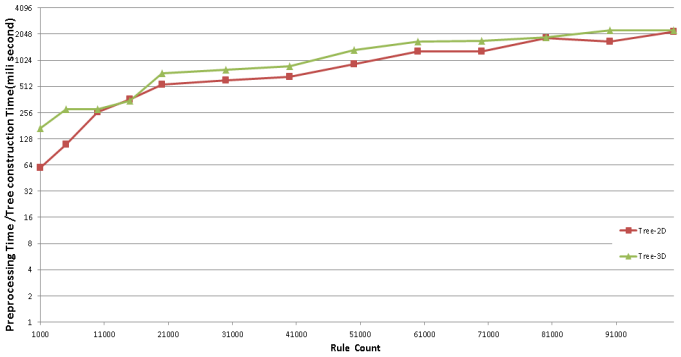

Figure-13
A comparison between the Non Linear (2D, 3D) algorithm as we increase rules to measure the preprocessing time or tree construction time

The preprocessing time or tree construction time (figure 13), shows that in case of the new algorithms preprocessing time/tree construction time are reasonable. In addition to the number of cuts, the dimension to cut at each internal decision tree node is also critical to the algorithm performance. A larger bucket size can help to reduce the size and depth of a decision tree, but it can induce a longer linear search time, so smaller bucket size has the counter-effects. By experiments we could determine the appropriate bucket size for the best tradeoff of storage and throughput. Generally, a larger bucket size means a worse search processing but this does not always hold.

The preprocessing time or the decision tree construction increases as the number of rules grows. Smaller bucket size requires significantly longer time to process. During experiments we found that the algorithm consistently demonstrates better performance and scalability on the worst case type rule sets with 20% of rules are repeated once and 10% of them are repeated twice, than on the other types of rule sets.

The decision tree algorithm supports incremental updates to some extent. To insert (or remove) a rule, we can just walk the decision tree similar to the search process using the rule specification. However, since the rules may overlap, the process requires updating on all the affected ones. Moreover, such

updates lead to a suboptimal decision tree which deteriorates performance. The algorithm implementers need to evaluate the impact of the preprocessing and incremental update cost for their particular applications.

The overall evaluation results are consistent with our expectations. Although the decision tree-based algorithm allows a nice tradeoff between storage and throughput, but the overall performance is not very promising. More study need to be done, at least, to find more systematic ways to fine-tune the configurable parameters, better adaptive decision-tree construction procedures and rule set structure. All the evaluation results should be normalized in a directly comparable way. In different applications, some criteria may be more important than others, but the evaluation should provide information and let readers make their own judgments. Because packet classification algorithms are mostly based on heuristics, different rule sets with different structures and sizes tend to give very different results.

Related Work

Packet classification continues to be an important challenge in network processing[1,4,6,13,15-17]. An excellent survey on Packet Classification Algorithms can be found in the survey and taxonomy of packet classification techniques paper[11].

Rule set intersecting is another technique, the key theme being that a partial rule match is easier than a full rule match, all at once. The packet header can be split into substring and matched with a subset of rules whose intersection will give rule to match the entire packet header. The Bit Vector (BV) algorithm and the aggregated bit vector (ABV) algorithm, represent the subset of rules for each partial match by using bit vectors[7,14] . Different methods can be used for partial header lookup. A direct lookup table can provide the fastest method even if it consumes more storage. Binary search and longest prefix matchingare considered as well-established single field look up techniques[18,19,20,21].

As has been already discussed in earlier part of our paper packet classification viewed geometrically uses idea on the construction of data structures and representation of rules. The preprocessing of rule sets uses the strategy of cutting or projecting of the multidimensional space. The basic strategy of "divide and conquer" is used, as the initial rule set is too large and cannot be handled under limited storage or time. The rule set is partitioned and regrouped, so that a packet can quickly identify a reduced rule set that includes the matching rule. Woo's modular packet classification, Multidimensional Cuttings (HyperCuts), and Hierarchical Intelligent Cuttings (HiCuts)use this approach in algorithms[4, 8, 9]. On smaller subsets parallel look ups are possible speeding up the classifications significantly[20,21].

Every technique possesses some limitation. To achieve good performance the algorithm has to be designed to combine all approaches, their best characteristics and also utilize well the time-space tradeoff.

Conclusion

An algorithm has been proposed, based on Recursive Dimensional Cutting (DimCut), with a decision tree structure. Proper implementation of the algorithm can help high performance packet classification to enable the routers, firewalls and security challenges in high-speed environments.

We tried to search for characteristics of real classifiers that can be exploited in pursuit of algorithms that are "fast enough" and use "not too much" space. We implemented the new algorithm on top of HiCuts by new heuristics and evaluated the proposed algorithm by measuring the packet classification Time, the number of packet per second processing, the search count, the Preprocessing Time/Tree Construction Time, the number of buckets (leaves), depth of the tree structure and Threshold. All of these have to be explicit constructs and search performance is evaluated by running through it large number of random packets that obey the weight specifications.

The proposed algorithm was analyzed for the different tradeoffs between the bucket size, number of cuts, number of rules, and number of levels. The SPSS statistical software analyses data while the Normality test that is done by Kolmogorov–Smirnov and the Difference test that is the Paired sample T Test, to check the final results. According to the results we draw several graphs for better representation.

Finally, we note that DimCut can easily be implemented in hardware at line speeds using a pipeline and on-chip SRAM. As the tree structure constructed by DimCut has levels no greater than 5, these needs only 5 pipeline levels, which is well within current hardware limits. We believe that DimCut can be a viable packet classification algorithm that gives the deterministic performance with flexibility for system designers to tradeoff the components. We believe this project will benefit the research and design community as a whole. Although, the decision tree algorithms allow a nice tradeoff between the storage and throughput, but the overall performance is still not very promising. More study need to be done, at least, to find more systematic ways of fine-tuning the configurable parameters, better adaptive decision-tree construction procedures and rule set structure.

References

1. Song H. and Turner J., Toward Advocacy-Free Evaluation of Packet Classification Algorithms, in *IEEE Transactions on Computers*, **60**, **(2011)**

2. Bauer M., Paranoidpenguin: Using Iptables for local security, in Linux Journal, Available at http://www.linuxjournal.com/article/609, August **(2002)**

3. Napier D., IPTables/NetFilter – Linux's next generation stateful packet filter, in Sys Admin - Security: *The Journal*

for UNIX Systems Administrators, **10(12),** 8, 10, 12, 14, 16, **(2001)**

4. Woo T., A Modular Approach to Packet Classification: Algorithms and Results, in Proceedings of INFOCOM 2000, Nineteenth Annual Joint Conference of the *IEEE* Computer and Communications Societies, **3**, 26-30 **(2000)**

5. Decasper D., Dittia Z., Pantlkar G. and Plattner Scottberg B., Router plugins: A software architecture for next generation routers, in Proceedings of ACM Sigcomm, 191-202, Vancouver, Canada, **(1998)**

6. Srinivasan V., Varghese G., Suri S. and Waldvogel M., Fast and scalable layer four switching, in Proceedings of ACM Sigcomm '98 , 191-202,Vancouver, Canada, **(1998)**

7. Stihdis D. and Lakslunan T.V., High-speed policy-based packet forwarding using efficient multi-dimensional range matching, in Proceedings of ACM Sigcomm, 203-214, Vancouver, Canada, August 31 - September 4 **(1998)**

8. Singh S., Baboescu F., Varghese G. and Wang J., Packet Classification using Multidimensional Cutting, in Proceedings of the ACM SIGCOMM '03 Conference on Applications, Tech., Archi., and Protocols for Computer Communication (SIGCOMM '03), 213 – 224 **(2003)**

9. Gupta P. and McKeown N., Packet Classification Using Hierarchical Intelligent Cuttings, in Proceedings of IEEE Symp. High Performance Interconnects (HotI), 7, **(1999)**

10. Vamanan B., Voskuilen G., Vijaykumar T.N., EffiCuts: optimizing packet classification for memory and throughput, in Proceedings of Proceedings of the ACM SIGCOMM 2010 conference on SIGCOMM, New Delhi, India, **(2010)**

11. Taylor D., Survey and Taxonomy of Packet Classification Techniques, in Proceedings of ACM Computing Surveys (CSUR), **37(3)**, 238 - 275 **(2005)**

12. Ahmadi M. and Wong, S., Modified Collision Packet Classification using Counting Bloom Filter in Tuple Space, in Proceedings of International Conference on Parallel and Distributed Computing and Networks (PDCN 2007), Innsbruck, Austria, **(2007)**

13. Gupta P. and McKeown N., Packet Classification on Multiple Fields, in proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication in ACM SIGCOMM '99, 147-160 **(1999)**

14. Baboescu F. and Varghese G., Scalable Packet Classification, in Proceedings of the ACM SIGCOMM, 199–210 **(2001)**

15. Feldmann A. and Muthukrishnan S., Tradeoffs for Packet Classification, in Proceedings of the IEEE INFOCOM, Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies, **3**, 1193–1202 **(2000)**

16. Waldvogel M., Varghese G., Turner J. and Plattner B., Scalable High Speed IP Routing Lookups, in Proceedings of the ACM SIGCOMM, 25-38 **(1997)**

17. Srinivasan V. and Varghese G., Fast Address Lookups Using Controlled Prefix Expansion, in Proceedings of the ACM Trans. Computer Systems, **17**, 1-40 **(1999)**

18. Eatherton W., Varghese G. and Dittia Z., Tree Bitmap: Hard- ware/Software IP Lookups with Incremental Updates, ACM SIGCOMM Computer Comm. Rev., **34(2)**, 97-122 **(2004)**

19. Song H., Turner J. and Lockwood J., Shape Shifting Tries for Faster IP Lookup, in Proceedings of the IEEE Int'l Conf. Network Protocols (ICNP'05), **(2005)**

20. Song H., Turner J. and Dharmapurikar S., Packet Classification Using Coarse-Grained Tuple Spaces, in Proceedings of the ACM/IEEE Symp, Architecture for Networking and Comm., Systems (ANCS '06), 41- 50 **(2006)**

21. Srinivasan V., Suri S. and Varghese G., Packet Classification Using Tuple Space Search, in Proceedings of the ACM SIGCOMM'99, 135-146 **(1999)**